Enhancing Structural Views of Software Systems by Dynamic Information

Stefan Voigt, Johannes Bohnet, Jürgen Döllner
Hasso-Plattner-Institute – University of Potsdam, Germany
{stefan.voigt, bohnet, doellner}@hpi.uni-potsdam.de

Abstract

Understanding software systems comprises the analysis of different aspects of the respective systems, such as dynamic and static analysis with all their facets. Consequently, developers study different kinds of information and create different mental models. We introduce a visualization technique that facilitates cross referencing mental models, in particular models that describe the structure and models describing the behavior of software systems. To achieve this goal, we enhance structural views by runtime information depending on the current focus of a sequential view. Animation enables developers to explore how the system's state changes over time, by this, supporting developers in understanding program behavior.

1. Introduction

Software visualization has a great potential to facilitate program comprehension by displaying artifacts and relations that are relevant for individual comprehension tasks. Typically, multiple views are used that display different aspects of software systems. This results in multiple mental models being created.

Comprehension can be improved by cross-referencing mental models [8] that result from different visualizations. Consequently, there is a need for facilitating the transition of findings between views, e.g. by synchronizing them [2]. Among others, the analysis of the system's structure can be supported by transitions to behavior views, as they show in which time ranges and execution phases the artifacts are active.

Using animation to visualize program executions may improve information bandwidth [1] of structural visualizations and can reduce the need for switching to a sequential visualization to understand the system behavior. Due to the possibly large size of execution traces, animations of program executions need to consider perceptive limits [7] and provide techniques to alleviate them [9].

In our previous work, we introduced an offline visualization concept that combines multiple views to help developers in locating and understanding features of complex software systems [3]. Here we extend the concept.

The main contribution of this paper is a visualization technique that supports developers in cross reference mental models, in particular, models that describe the structure and models describing the behavior of software systems. To reach this goal, we enhance a structural visualization by information that describes the behavior of the displayed artifacts. This information represents a) the point in the system's execution that is focused by the sequential view, b) the history of execution to that point and c) the future of execution to that point. Consequently, both views are synchronized and display always the same point time.

Based on the additional information depicted in the structural view, we introduce navigation concepts that support developers in exploring program behavior. Among others, developers can study the behavior of the artifacts displayed in the structural view by animating system executions. The animation concept is designed with respect to the human perception mechanisms and allows developers to observe execution over time even at a high animation rate.

The visualization technique is implemented and integrated into the CGA-Framework [3] for analyzing complex C/C++ software systems.

2. Visualization Concept

First, we introduce both the sequential and the structural view. Afterwards we describe the concept of enhancing the structural view by dynamic information that is synchronized with the dynamic view.

978-1-4244-5026-8/09/$25.00 ©2009 IEEE

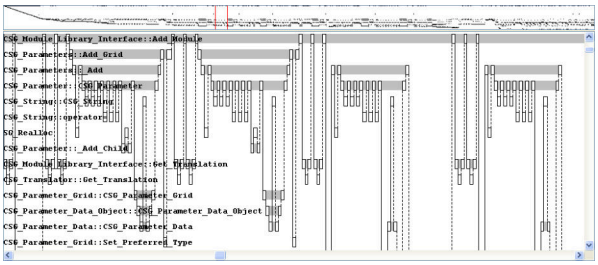

Figure 1: Sequential view

The sequential view (Figure 1) lists methods vertically and maps the time horizontally. The view displays method executions as horizontal bars. By that, it visually depicts when executed functions are active (white bar) and when they initiated a call and are waiting to receive back control (gray bar). The view provides zooming and panning to support the exploration of traces at different levels of granularity. The vertical alignment of the methods is optimized to fit best the currently displayed time window. The sequential view is supported by a coarse-grained view that facilitates detecting different phases of execution (Figure 1, on the top).

The structural view (Figure 2) displays a subset of methods and call relations of software systems under study. It helps developers to analyze the role of methods in the focus of interest by depicting its *surroundings* as a clustered call graph. The surroundings comprise

- methods that are closely related to the focused method by means of call distance;
- call relationships between these methods;
- the hierarchy of implementation units the methods belong too, e.g., files and directories.

2.1. Integrating Dynamic Information in Structural Views

We enhance the structural view by displaying dynamic information that relates to the point in time that is focused by the sequential view. Both views are synchronized by means of displaying always the same point in time. The dynamic information helps to understand the role of the artifacts displayed in the structural view. Among others, the following questions can be answered within the structural view:

- Which methods have already been executed?
- Which methods will be executed?
- What are the next/previous active methods?
- Which methods are currently executed?

We address the last question by displaying the current call stack within the structural view. The stack is depicted using a color gradient from black to light red, whereas the black artifact forms the bottom of the stack (Figure 2b).

To answer the other questions, we combine two concepts. Firstly, node-colors encode how often the methods have been executed with regard to the focused time. If all activities of a node are in the future (with respect to the focused point in time) it is colored dark gray. If all executions are in the past, it is colored white. All other nodes are colored light-gray. (In black/white print outs the technique interferes with the call-stack visualization. For that reason, it is not displayed in the screenshots.)

Secondly, nodes display embedded timelines, which depict events that relate to the artifact they represent (Figure 2). The timeline distinguishes entry events (1) and exit events (2). Arrows indicate that relevant events are outside the visible time range (3).

To reduce the cognitive load on developers when analyzing the structural view, the visualization comprises a concept of intelligent labeling. The labels are filtered with regard to the time window displayed in the sequential view. That is, only labels of artifacts are displayed whose activities are related to the currently analyzed point in time.

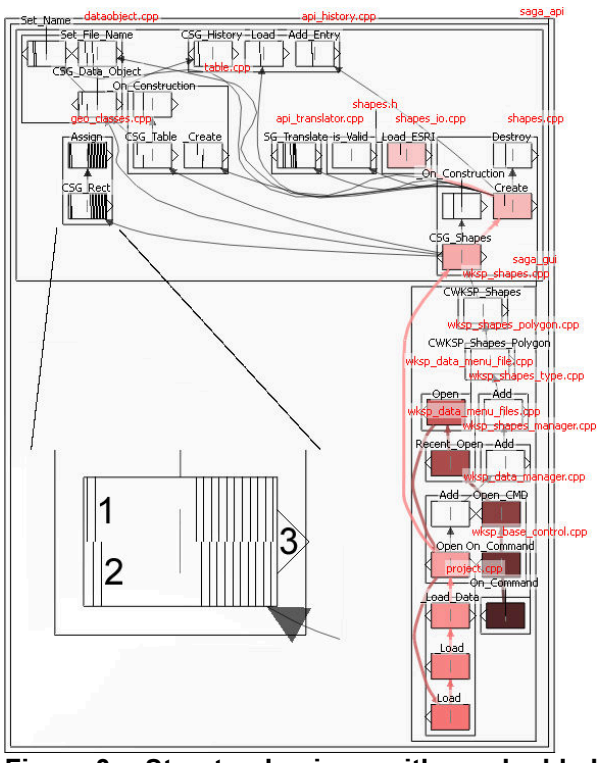

Figure 2: Structural view with embedded timelines and call stack visualization.

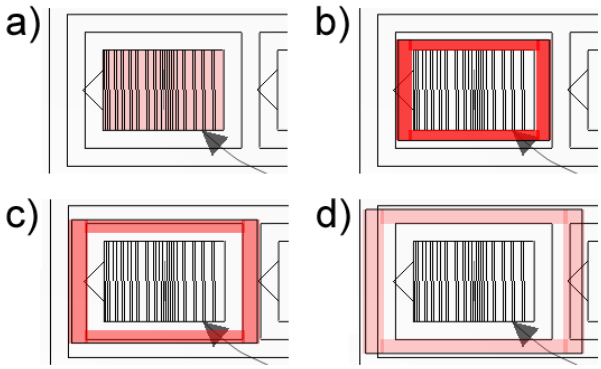

Figure 3: Afterglow technique. a) Method on stack. b) - d) Method is taken from stack, resulting in an afterglow silhouette that increases in size and fades out.

4. Navigation Concept

Users either use the coarse-grained sequential view to navigate to arbitrary points in time of the captured execution or they use mouse-dragging on the finer grained sequential view. The embedded timelines are used to focus events. Clicking on them displays other activities of the method. The arrow is used to navigate to the next event that is currently out of sight. This supports the exploration of method activities that are spread over the trace.

4.1. Animation

Program executions are animated in the structural view by sliding the focused point in time continuously. This allows developers to study system behavior within the structural view. Our implementation provides multiple kinds of interaction. Among others, animation may be stopped or adjusted in speed.

The animation rate is critical for comprehension. If the animation is too fast, the events might not be perceived correctly [9]. If it is too slow, animating large sections of traces consumes too much time. Additionally there are cognitive limitations in perceiving long sequences of events [7].

Short method executions might not be visually perceived. To enhance the perception, we introduce a technique named *afterglow*, which highlights the silhouette of nodes for a certain amount of time after the related methods finished execution. As depicted in Figure 3, the silhouette increases in size and decreases in intensity to encode the time that is passed since the method finished execution.

If users navigate to other points in time, the afterglow silhouette shows which methods are executed within the time range that is bridged by the navigation. This helps in understanding which methods are active in certain execution phases.

5. Discussion

The visualization concept allows developers to explore both, the structure and the behavior of complex software systems. In the following we discuss limitations of our approach.

Scalability: Encoding the call stack as a color gradient limits the scalability by means of the stack size. Consequently, increasing stack size results in decreasing color differences. Depending on the positions of the elements in the graph, i.e., the layout, their correct stack-order might not be perceived preattentively. Additionally, the recursion depth is not explicitly visualized.

A second scalability aspect relates to the size of the graph. If it requires zooming, e.g., for reasons of labeling, part of the animation can take place in areas that are currently not visible.

Accuracy: The visualization technique comprises concepts to identify executed methods at high rate. However, at high rate, the order of executions might not be perceived accurately. Consequently, for accurate order perception, the animation speed needs to be decreased. Additionally, if time ranges are skipped, the intensity of the afterglow-silhouette depends on the number of skipped executions. Especially if many executions have been skipped, their accurate count cannot be perceived.

Multithreading: As the visualization technique encodes 'the call stack', the approach is limited to the analysis of one thread per time.

6. Related work

Jinsight [4] is a tool for visualizing dynamic system information. It combines multiple synchronized visualizations and provides navigation between them. The structural view display cumulated dynamic information such as the number of calls. Our approach is different in that we integrate information that relates to a particular point in the system's execution.

Walker et al. [10] integrate dynamic information in a structural high level view using a sequence of cels. Each cel represents a particular point in the system's execution and displays current and historical information to that point. They provide concepts of animation and interaction. Their approach differs

from ours in that we provide additional interaction concepts and synchronization with a sequential view.

Greevy et al. [6] visualize behavior of object oriented software systems in 3D. Their approach displays classes and inheritance relationships on the 2D ground plane. For arbitrary points in execution, active instances are displayed in the third dimension as boxes on top of classes and active messages are displayed as edges between the instances. The visualization enables developers to step through the events of execution traces, but in contrast to our approach it only informs about one point in time at a moment.

Deelen et al. [5] visualize dynamic program aspects using a structural and a sequential view. The structural view displays classes and communication. For arbitrary points in time within execution, the view encodes metrics such as the number of active instances. Program execution is simulated by means of a token that is passed along the arcs displaying how messages are exchanged between classes. In contrast to our approach, they focus on class interactions and do not provide concepts that facilitate the perception of long sequences of message passes.

7. Conclusions

In this paper, we described a visualization technique that allows developers to explore the structure and the behavior of complex software systems. The technique comprises concepts to support developers in cross referencing different mental models on software systems. This goal is achieved by synchronizing structural and sequential views. In particular, relating on the focus of the sequential view, the structural view depicts the current state of the program, that is, it shows the call stack and embeds timelines to inform on the previous and next activities of the displayed artifacts. Execution is animated to support developers in understanding the behavior of displayed artifacts. Additionally, the visualization technique comprises navigation methods and concepts to facilitate the perception of animations of the execution at a high rate.

Future Work: Sequential visualizations provide different techniques to handle large execution traces [2]. Among others, they allow to visually collapse reoccurring patterns such as loops to reduce visual complexity. An interesting research topic is to apply these concepts to animation. Another interesting research topic is the application of our approach to multithreaded programs.

References

[1] L. R. Bartram, “Perceptual and Interpretative Properties of Motion for Information Visualization”, In *Proceedings of the 1997 Workshop on New Paradigms in Information Visualization and Manipulation*, ACM, 1997, pp. 3-7.

[2] C. Bennett, D. Myers, M.-A. D. Storey, D. M. German, D. Ouellet, M. Salois, and P. Charland, ”A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams”, In *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley, 20, 2008, pp. 291–315.

[3] J. Bohnet, S. Voigt, and, J. Döllner, „Locating and Understanding Features of Complex Software Systems by Synchronizing Time-, Collaboration- and Code-Focused Views on Execution Traces “, In *Proceedings of the 16th International Conference on Program Comprehension*, IEEE, 2008, pp. 268-271.

[4] W. De Pauw, E. Jensen, and N. Mitchell, “Visualizing the Execution of Java Programs”, In *Lecture Notes In Computer Science*, Springer-Verlag, vol. 2269, 2002, pp. 151-162.

[5] P. Deelen, F. van Ham, C. Huizing, and H. van de Watering, “Visualization of Dynamic Program Aspects”, In *Proceedings of the 4th International Workshop on Visualizing Software for Understanding and Analysis*, IEEE, June 2007, pp. 39-46.

[6] O. Greevy, M. Lanza, and C. Wysseier, “Visualizing live software systems in 3d”, *In Proceedings of the Symposium on Software Visualization*, ACM, 2006, pp. 47-56.

[7] R. N. A. Henson, “Short-Term Memory for Serial Order: The Start-End Model”, *Cognitive Psychology*, Elsevier, 36(2), 1998, pp. 73-137.

[8] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller, “Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization”, In *Proceedings of the 5th International Workshop on Program Comprehension*, IEEE, 1997, pp. 17-28.

[9] B. Tversky, J. B. Morrison, and M. Betrancourt, “Animation: can it facilitate?”, *International Journal of Human-Computer Studies*, Academic Press, 57(4), 2002, pp. 247-262.

[10] R. J. Walker, G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak, “Visualizing Dynamic Software System Information through High-Level Models”, In *Proceedings of the 13th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications*, ACM, 1998, pp. 271-283.